German RLC

German Record Linkage Center
WORKING PAPER SERIES
NO. WP-GRLC-2012-01 | JULY 2, 2012

TDGen: A Test Data Generator for Evaluating Record Linkage Methods

Tobias Bachteler | Jörg Reiher

TDGen: A Test Data Generator for Evaluating Record Linkage Methods

Tobias Bachteler and Jörg Reiher

German Record Linkage Center
University of Duisburg-Essen, D-47057 Duisburg, Germany
recordlinkage@iab.de, http://www.record-linkage.de

July 2, 2012

1 Introduction

The record linkage field is notoriously short of appropriate test data for methodological research on linkage algorithms and procedures. Since there are few real world data sets publicly available, research on record linkage methodology must often be based on artificially generated test decks. At present, most studies are based on the data set generator implemented as part of the Febrl data linkage system [1], [2]. The Febrl data set generator constitutes an outstanding contribution to record linkage research because it provided the community with a tool for generating artificial test data far more complex and elaborated than previous simulators as DBGen[1].
However, in the course of our research on record linkage methods we realized our need for some additional features and modifications of the Febrl generator. Particularly, we wanted the generator to be able to take an arbitrary test file and generate a second, garbeld version of it and to allow more flexible control of the error insertion probabilities. Since the Febrl generator is available under an open software license, we initially thought about modifying it. However, since this meant to set up an essentially new structure and logic, we decided to design and built a completely new test data set generator.
Our record linkage test data generator, TDGen for short, is designed to take a test data file and to provide a garbled version of it by introducing simulated errors. We provide TDGen as an extension to the information miner KNIME [4].
We drew heavily on the excellent work of Peter Christen and his colleagues [1], [2] regarding the error types TDGen provides. Specifically, we adopted most of its generic error types, the concept of files containing misspelled variants of strings, the way OCR-type errors are introduced, the idea of introducing phonetic errors by reverse-modelling their encoding rules, and the way typographical errors are inserted from the Febrl data simulator.

2 The principles of the test data generator

We designed TDGen according to a three basic principles. First, TDGen should allow the user to put in a test data file containing typical record linkage identifiers and to generate a second, garbled version of this file by introducing simulated errors. Second, TDGen should allow the user to specify as flexibly as possible the degree and type of error introduced. Third, users should be

[1] The first version of DBGen was provided by Mauricio Hernandez and used in [3]. Its source code can be obtained via the RIDDLE Repository (http://www.cs.utexas.edu/users/ml/riddle/index.html). Amit Chandel and Mohammad Sadoghi (University of Toronto) provided an update which can be downloaded at http://queens.db.toronto.edu/~mo/spider/index.php#data.

able to adapt, modify and extend TDGen requiring a minimum of programming skills.

We provide TDGen as a KNIME extension along with a readily working build-up within KNIME, a so called KNIME workflow. This workflow produces from an input file a garbled version as an output file. This report describes the installation and usage of this basic TDGen workflow. We assume that the user at first will adapt this workflow by configuring the basic workflow according to his needs and by adding some generic KNIME nodes. Later on, the user can assemble his own TDGen workflow from the nodes delivered with our KNIME extension and program his own TDGen nodes as well.

TDGen allows to configure the error-generating process for rows, columns and cells separately. First, the user defines the fraction of erroneous rows, then the number of erroneous columns in those rows and finally the number of errors in the concerned cells.

The basis of the cell error assignment is concept of the *error scores*. For each data cell an *error score limit* is defined during the configuration process. If a certain error type is realized within a data cell, its associated *error type score* will be added to the data cell's *error score*. As long as the *error score limit* is not reached, additional errors can be realized in a data cell. On the other hand, if the *error score limit* is reached or exceeded, no further errors can be realized. Since the user can set the *error type score* of each error type, he has good control over the error formation process in the data cells.

3 Installation

3.1 32 bit Windows

1. Download the self-extracting archive from "http://knime.org/downloads/knime/win32exe".
2. Extract to "C:/Program Files". This will create "C:/Program Files/knime_2.5.X".
3. Unzip the archive "simulator_data.zip" to "C:/". This will create "C:/simulator_data".
4. Copy all .jar files in "C:/simulator_data" to "C:/Program Files/knime_2.5.X/dropins".
5. Start "cmd.exe", move to "C:/Program Files/knime_2.5.X", and execute the command "knime -clean".
6. In the window "Workspace launcher", click on "ok".
7. Click on "Open KNIME workbench".
8. Choose "Import KNIME workflow ..." from the menu "File".
9. Select "Select archive file" in the panel "Source:".
10. Via the Browse-button choose "tdgen-basic-workflow.zip" from the folder "C:/simulator_data".
11. Open the TDGen-workflow by a double-click on "tdgen-basic" within the tab "Workflow Projects".

    Note that from now on KNIME can be started regularly.

3.2 32 bit Linux

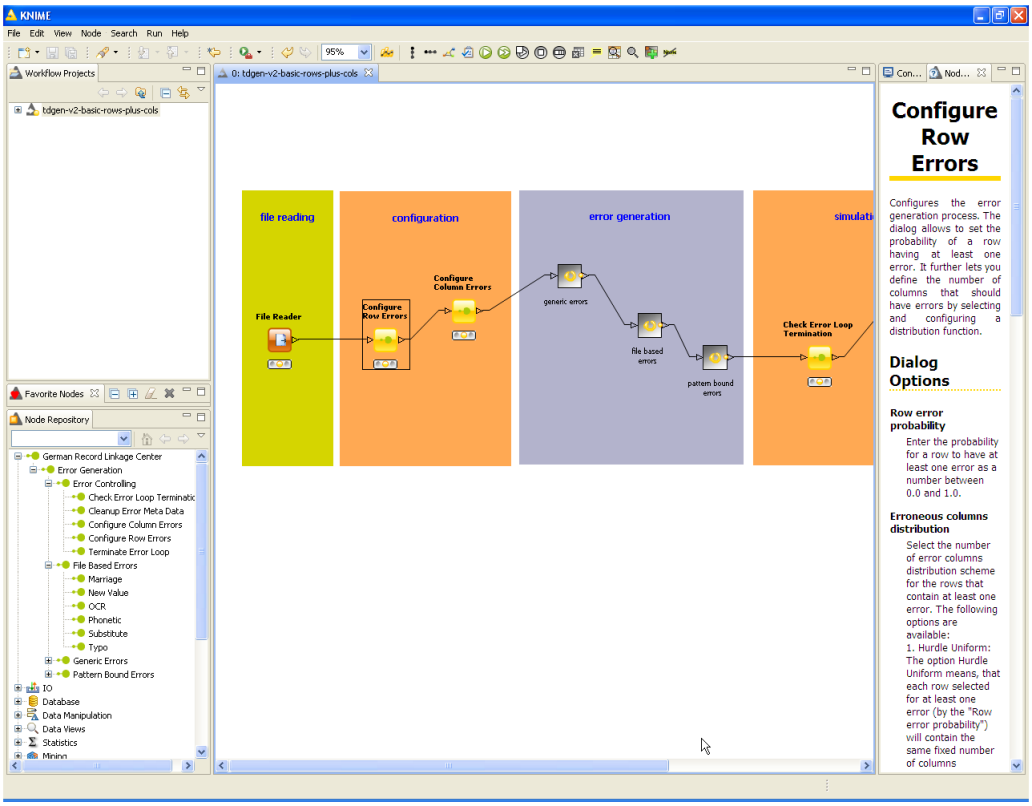
1. Download the self-extracting archive from "http://knime.org/downloads/knime/linux32".
2. Extract to your home directory "/home/<username>". This will create "/home/<username>/knime_2.5.X".
3. Unzip the archive "simulator_data.zip" to "/home/<username>". This will create "/home/<username>/simulator_data".

4. Copy all *.jar files in “/home/<username>/simulator_data” to “/home/<username>/knime_2.5.X/dropins”.
5. Start the console, move to “/home/<username>/knime_2.5.X”, and execute the command “./knime -clean”.
6. In the window “Workspace launcher”, click on “ok”.
7. Click on “Open KNIME workbench”.
8. Choose “Import KNIME workflow ...” from the menu “File”.
9. Select “Select archive file” in the panel “Source:”.
10. Via the Browse-button choose “tdgen-basic-workflow.zip” from the folder “/home/<username>/simulator_data”.
11. Open the TDGen-workflow by a double-click on “tdgen-basic” within the tab “Workflow Projects”.

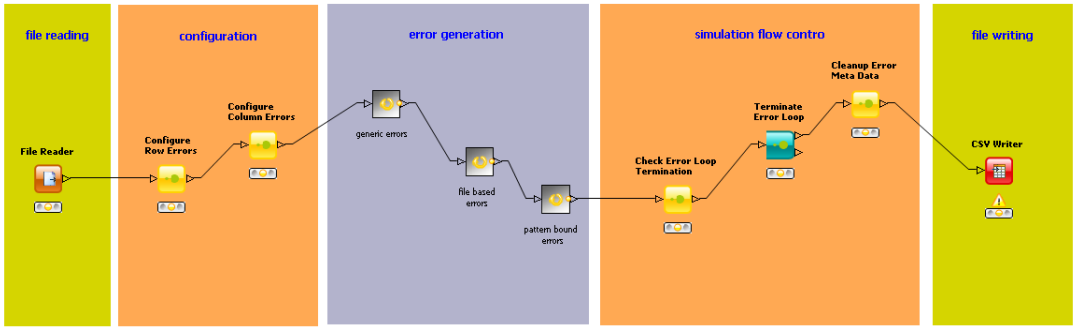
    Note that from now on KNIME can be started regularly.

4 Description and usage of the test data generator

After the successful import of the basic TDGen workflow, KNIME’s GUI should look as this:



In the Node Repository there is now an additional entry “German Record Linkage Center”. There you find all of TDGen’s nodes along with a detailed description.



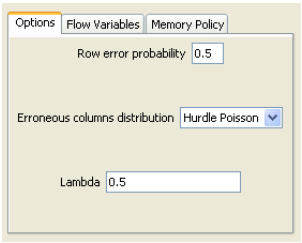
4.1 File reading

The input data file is feed in the workflow via the node "File reader". The user can configure KNIME nodes by right-clicking on the node and choosing "Configure...". The node "File reader" is a generic KNIME node. Please consult its node description for help.

In our basic workflow we read the file "C:/simulator_data/sample_data_500.csv", a tab-separated, utf-8 coded data file containing person data with typical record linkage identifiers. Please make sure to choose utf-8 in the tab "Character decoding" after a click on the button "Advanced...". There is also a second sample input data file "sample_data_10000.csv". Please change the "File reader" node to put in this larger sample file.
Note that TDGen expects string attributes, so be sure to configure the node accordingly when reading your own data.

4.2 Configurating the error generation process

The error generation is configurated by the two nodes "Configure Row Errors" and "Configure Column Errors".
When opened, the configuration dialogue of the nodes "Configure Row Errors" looks as this:



Specifying the fraction of erroneous rows The input in the field "Row error probability" specifies the probability of each row containing at least one error.

Specifying the number of erroneous columns The distribution of erroneous columns determines how many columns will be error-prone in each erroneous row. There are three alternatives:

1. **Hurdle Poisson**
   If you choose the option “Hurdle Poisson”, you have to specify the Poisson mean via the box “Lambda”. Then, the number of columns containing errors will be sampled from the specified Poisson distribution with the row error probability as hurdle (i. e. as the proportion of zeros) [5].
2. **Hurdle Uniform**
   If you choose the option “Hurdle Uniform”, each erroneous row will have the same number of error columns. You have to specify this number in the field “Error columns per row”.
3. **Hurdle Manual**
   The option “Hurdle Manual” lets you manually enter the relative frequencies of one, two or more error columns per row. Please consider the node description for syntax details.

When opened, the configuration dialogue of the nodes “Configure Column Errors” looks as this:

Selecting attributes On the left is the list of the attributes from the input file. The check box selects an attribute for error insertion.

Specifying the relative error frequency for the attributes As a result of the “Row error configuration” process, for each row the number of erroneous columns is defined, but not yet which columns shall be chosen for this. By the ”Error frequency” fields you can specify the chance that the respective column is being selected. Of course, once an attribute is chosen to be erroneous in the concerned row, it is removed from the select list for that row.

Selecting an error score limit distribution The number of errors within a cell is specified via the drop-down menu “Error distribution”. There are three alternative *error score limit* distributions to choose from, Hurdle Uniform, Poisson, and Hurdle Poisson.

1. **Hurdle Uniform**
   The option Hurdle Uniform means, that errors will be inserted into a certain fraction of the data cells (defined via the box "Error probability") only and there will be the same error score limit for each cell selected for error insertion (defined via the box "Global Score").
2. **Poisson**
   If you choose the option "Poisson", you have to specify the Poisson mean via the box "Lambda". Then, the *error score limit* for each cell will be sampled from the specified Poisson distribution.
3. **Hurdle Poisson**
   As compared to the "Poisson" option you can additionally specify the proportion of cells containing errors via the box "Error probability". The Poisson sampling will only be applied within the cells selected for error insertion.

Selecting the error types In the box "Errors" you specify the error types that might be realized within the data cells of the attribute. The error types are selected by their error type shortcuts. The relative probabilities of the error types are defined by the relative magnitude of the associated numbers.

For example, consider the following showcase entries into the "Errors" box:

Example A: abb=0.5,dtd=0.5,pho=1.0,typ=1.0,ocr=1.0
Example B: pho=40,typ=40,ocr=40

The probability of an individual error type equals its number divided by the sum of the all numbers. In example A, the probability of an abbreviation error (shortcut abb) would be 0.5/4 = 1/8. In example B, the probability of a typographical error (shortcut typo) would be 40/120 = 1/3.

The error type shortcuts can be learned from the lists in the Appendix or the node descriptions of the error types in TDGen.

4.3 Error generation

In the error generation process, each error type is represented by an individual node. For each data cell the realization of an error type is determined according to the simulation flow configuration.

We distinguish three broad classes of error types: Generic errors are readily applicable to an attribute without requiring additional files. File based errors require specialised substitution lists or other additional files. Pattern bound errors expect the attribute coming with a certain pattern (for example a 5 digit number). The error types are listed in the Appendix of this report. Detailed descriptions and explanations of the individual error types can also be found in the respective node descriptions in TDGen.

In the basic TDGen workflow, the individual error nodes are grouped in KNIME meta nodes. The content of a meta node can be viewed by a double click on the meta node.

4.4 Simulation flow control

The simulation flow control is comprised of three nodes. The "Check Error Loop Termination" node calculates the number of rows that need further error processing and passes the result to the "Terminate Error Loop" node. Based on the passed result the "Terminate Error Loop" node decides whether the error generation should be continued or be terminated: If the number of rows that need further error processing is larger than zero, the loop is executed again. Otherwise the generated data table including internal meta data columns for the error generation process is passed to the "Cleanup Error Meta Data" node. The node "Cleanup Error Meta Data" removes the internal meta data columns from the "Terminate Error Loop" output table.
The "Check Error Loop Termination" node can be configured to enforce a termination of the error generation loop after a given number of loops. If this maximum number of loops is reached, then the node will force the "Terminate Error Loop" to stop the error generation process by passing zero as the number of remaining error rows to the termination node, even if there are still rows left that need error processing. This leads to a termination of the process even if for some rows the error score limit cannot be reached. For example because the error types set for a column aren't applicable for a particular cell. The default maximum value of 20 loops should be sufficient for any realistic error generation scenario and should only be changed if the number of errors for a single cell can exceed 20.

4.5 File writing

The garbled version of the input file is written by the generic KNIME node "csv writer". Please consider its node description to learn about its usage.

4.6 Starting the data generation process

Before starting a TDGen run, you should reset the whole workflow by choosing "Select all" from the "Edit" menu (or Cntrl+A) and then choosing "Reset" from the "Node" menu (or F8) to grasp any changes in the workflow.

To start the data generation process do a right click on the "csv writer" node in the "file writing" area and choose "Execute".

5 Adapting and modifying TDGen

You can adapt the basic TDGen workflow by supplying your own input file, by configurating the error process according to your needs and by supplying your own additional files to the file based error types. For example, if you want to adapt the typo node to a non German keyboard, you can supply a modified "typo_example.csv".

Modifying the basic TDGen workflow is easy using KNIME's node repository. For example, the "Database Manipulation" nodes allow you to sample, filter or split your input file according to your individual needs.

References

1. Peter Christen. Probabilistic data generation for deduplication and data linkage. In: Marcus Gallagher, James M. Hogan, and Frederic Maire (eds.) *Proceedings of the 6th International Conference on*

Intelligent Data Engineering and Automated Learning: 6–8 July 2005; Brisbane, pp. 109–116, Berlin: Springer 2005.

2. Peter Christen and Angus Pudjijono. Accurate synthetic generation of realistic personal information. In: Thanaruk Theeramunkong, Boonserm Kijsirikul, Nick Cercone, and Tu-Bao Ho (eds.), *Advances in Knowledge Discovery and Data Mining. Proceedings of the 13th Pacific-Asia Conference: 27–30 April 2009; Bangkok*, pp. 507–514, Berlin: Springer 2009.
3. Mauricio A. Hernández and Salvatore S. Stolfo. Real-world data is dirty: data cleansing and the merge/purge problem. *Data Mining and Knowledge Discovery*, 2(1):9–37, 1998.
4. Michael R. Berthold, Nicolas Cebron, Fabian Dill, Thomas R. Gabriel, Tobias Kötter, Thorsten Meinl, Peter Ohl, Christoph Sieb, Kilian Thiel, and Bernd Wiswedel. KNIME: The Konstanz information miner. In: Christine Preisach, Hans Burkhardt, Lars Schmidt-Thieme, and Reinhold Decker (eds.) *Data Analysis, Machine Learning and Applications*, pp. 319–326. Berlin: Springer 2008.
5. Norman L. Johnson, Adrienne W. Kemp, and Samuel Kotz. *Univariate Discrete Distributions*. 3rd ed., Hoboken, NJ: Wiley 2005.

Appendix

A1. Generic Errors

1. Abbreviate
   Randomly replaces one (or all depending on the configuration) token by its first letter and a dot (meaning its abbreviation). Tokens are identified by splitting the input at the blank characters.
   Example: Michael → M.
   Shortcut: abb
2. Blanks To Hyphens
   Replaces any occurrence of a blank by hyphens ("-").
   Example: New Haven → New-Haven
   Shortcut: bth
3. Delete Blank
   Randomly removes one blank from the input if there is any. Returns input otherwise.
   Example: New Haven → NewHaven
   Shortcut: deb
4. Delete Token
   Randomly removes one token from the input if there is more than one token. Tokens are identified by splitting the input at the blank characters.
   Example: New Haven → Haven
   Shortcut: det
5. Digit To Dot
   Randomly replaces one digit by a dot (".") if there is any digit in the input.
   Example: 56th street → 5.th street
   Shortcut: dtd
6. Drop Last Character
   Removes last character from the input.
   Example: Michael → Michae
   Shortcut: dlc
7. Drop Last Two Characters
   Removes the last two characters from the input.
   Example: Michael → Micha
   Shortcut: dlt
8. Drop Nondigits
   Removes all occurrences of nondigits from input.
   Example: 56th street → 56
   Shortcut: dnd
9. Hyphens To Blanks
   Replaces any occurrence of a hyphen "-" by blanks.
   Example: Jean-Francois → Jean Francois
   Shortcut: htb
10. Insert Blank
    Randomly inserts one blank into the input.
    Example: Michael → Mi chael
    Shortcut: inb
11. Keep First Token
    Keeps only the first token of the input. Tokens are identified by splitting the input at the blank characters.

Example: New Haven → New
Shortcut: kft

12. Keep Longest Token
    Keeps only the longest token of the input. Tokens are identified by splitting the input at the blank characters.
    Example: New Haven → Haven
    Shortcut: klt
13. Replace By Empty String
    Replaces the input by an empty string.
    Example: Michael →
    Shortcut: rbe
14. Resolve Umlauts And Sharp S
    Replaces any occurrence of a German umlaut as well as the German sharp S by its ASCII representations
    Example: Sträßer → Straesser
    Shortcut: rus
15. Revert Resolved Umlauts And Sharp S
    Replaces any occurrence of a German umlaut replacement as well as a "ss" by its umlaut / sharp S representations
    Example: Straesser → Sträßer
    Shortcut: rru
16. Special Characters To Blanks
    Replaces any occurrence of a character that isn't either a digit or a German letter (US letters + umlauts + sharp S) by blanks.
    Example: Lord-Curzon-Street 56/A → Lord Curzon Street 56 A
    Shortcut: scb
17. Swap Fields
    Swaps the values of 2 columns.
    Example: Michael; Smith; Curzon St; New Haven → Curzon St; Smith; Michael; New Haven
    Shortcut: swf
18. Swap Tokens
    Swaps two randomly chosen tokens of the input. Tokens are identified by splitting the input at the blank characters.
    Example: Lord Curzon Street → Lord Street Curzon
    Shortcut: swt

A2. File Based Errors

1. Marriage
   For usage with a surname column only. Either prefixes the original surname with an alternative surname and a hyphen or replaces the original surname. Each operation is chosen with equal probability 0.5. The alternative surname is randomly chosen from a given reference table. The operation is applied only if an appropriate column flags the row as representing a female and another column flags the row as representing a person of 18 years or older.
   Examples: Miller -> Smith-Miller; Miller -> Brown
   Shortcut: mar
2. New Value
   Replaces the given cell value by a randomly chosen new value from specified columns of a given reference table.
   Shortcut: new
3. OCR
   Insert simulated OCR errors based on a replacement frequency map. First the position of the OCR error within the string is chosen randomly with equal probability. Then all replacements and their frequencies for the chosen error position are fetched from the replacement frequency map. The replacement is chosen randomly from the fetched replacements weighted by the given frequencies. See example below.
   Shortcut: ocr
4. Phonetic
   Insert simulated phonetic errors based on a regular expression pattern replacement mapping. First the position of an error operation within the string is chosen randomly with equal probability. Then a source pattern that is applicable at the chosen error position is randomly chosen either from the internally or externally provided source-pattern-replacement map. The chosen pattern is replaced randomly by one of the provided replacement strings. If no replacement string is provided for the source pattern, then instead of a replacement the pattern will be removed from the input string.
   Shortcut: pho
5. Substitute
   Substitutes strings by a substitution value taken from one or more mapping files. If there are several substitution values, the replacement is randomly chosen.
   Shortcut: sub
6. Typo
   Insert simulated typographical errors. There are four typographical errors: Insertion, Deletion, Substitution, and Transposition of a character. The position of an error operation within the string is determined by a normal distribution with mean corresponding to the middle character of the string. The logic of this node corresponds to the mechanism used in the Febrl data set generator. If multiple alphabets are provided they will be concatenated. If there are multiple column transition rules for the same source key, only the last one is used. If there are multiple row transition rules for the same source key, only the last one is used.
   Shortcut: typ

A3. Pattern Bound Errors

A3.1 Date of Birth Errors

1. Alter Day
   Alters the day of a given date between 1–5 days up or down (probability for n days: $p = n^{\frac{1}{2}}$). Expects date to be formatted YYYY-MM-DD. If max/min is reached, day wraps around (31 → 1, 1 → 31) without changing the month or year of the date.
   Shortcut: sda
2. Alter Decade
   Alters the year of a given date between 1–5 decades up or down (probability for n decades: $p = n^{\frac{1}{2}}$). Expects date to be formatted YYYY-MM-DD. Stops when century boundary is reached.
   Shortcut: sde
3. Alter Month
   Alters the month of a given date between 1–5 months up or down (probability for n months: $p = n^{\frac{1}{2}}$). Expects date to be formatted YYYY-MM-DD. If max/min month is reached, month wraps around (12 → 1, 1 → 12) without changing the year of the date.
   Shortcut: smo
4. Alter Year
   Alters the year of a given date between 1–5 years up or down (probability for n years: $p = n^{\frac{1}{2}}$). Expects date to be formatted YYYY-MM-DD. Stops when decade boundary is reached.
   Shortcut: sye
5. Drop Century
   Removes the century from the year of a given date: Expects date to be formatted THDY-MM-DD and returns DY-MM-DD.
   Shortcut: dce
6. Drop Day
   Removes the day from a given date: Expects date to be formatted YYYY-MM-DD and returns YYYY-MM-.
   Shortcut: dda
7. Drop Leading Zero From Day
   Removes the leading zero from the day of a given date: Expects date to be formatted YYYY-MM-0D and returns YYYY-MM-D.
   Shortcut: dld
8. Drop Leading Zero From Month
   Removes the leading zero from the month of a given date: Expects date to be formatted YYYY-0M-DD and returns YYYY-M-DD.
   Shortcut: dlm
9. Drop Month
   Removes the month from a given date: Expects date to be formatted YYYY-MM-DD and returns YYYY–DD.
   Shortcut: dmo
10. Drop Year
    Removes the year from a given date: Expects date to be formatted YYYY-MM-DD and returns --MM-DD.
    Shortcut: dye
11. Drop Year And Decade
    Removes the year and decade from the year of a given date: Expects date to be formatted THDY-MM-DD and returns TH-MM-DD.

Shortcut: dyd

12. Reset Day
    Sets the day of a given date to 01: Expects date to be formatted YYYY-MM-DD and returns YYYY-MM-01.
    Shortcut: rda
13. Reset Month
    Sets the month of a given date to 01: Expects date to be formatted YYYY-MM-DD and returns YYYY-01-DD.
    Shortcut: rmo
14. Reset Year
    Sets the single digit of the year of a given date to 5: Expects date to be formatted YYYY-MM-DD and returns YYY5-MM-DD.
    Shortcut: rye
15. Round To Next Decade
    Rounds the year of a given date to the next decade: Expects date to be formatted YYYY-MM-DD.
    Shortcut: rnd
16. Swap Day And Month
    Swaps the day and the month of a given date. Expects date to be formatted YYYY-MM-DD and returns YYYY-DD-MM.
    Shortcut: sdm
17. Swap Day's Digits
    Swaps the digits of the day of a given date. Expects date to be formatted YYYY-MM-dD and returns YYYY-MM-Dd.
    Shortcut: swd
18. Swap Month's Digits
    Swaps the digits of the month of a given date. Expects date to be formatted YYYY-Mm-DD and returns YYYY-mM-DD.
    Shortcut: swm
19. Swap Year's First Two Digits
    Swaps the first two digits of the year of a given date. Expects date to be formatted yzYY-MM-DD and returns zyYY-MM-DD.
    Shortcut: syf
20. Swap Year's Last Two Digits
    Swaps the last two digits of the year of a given date. Expects date to be formatted YYyz-MM-DD and returns YYzy-MM-DD.
    Shortcut: syl

A3.2 German ZIP Code Errors

1. Alter 1st Digit
   Alters the 1st digit of the given German ZIP code randomly by $n = 1, 2, 3$ up or down (probability for alter n: $p = n^{\frac{1}{2}}$). Expects a 5 digit German ZIP code string input.
   Shortcut: s1d
2. Alter 2nd Digit
   Alters the 2nd digit of the given German ZIP code randomly by $n = 1, 2, 3$ up or down (probability for alter n: $p = n^{\frac{1}{2}}$). Expects a 5 digit German ZIP code string input.
   Shortcut: s2d
3. Alter 3rd Digit
   Alters the 3rd digit of the given German ZIP code randomly by $n = 1, 2, 3$ up or down (probability for alter n: $p = n^{\frac{1}{2}}$). Expects a 5 digit German ZIP code string input.
   Shortcut: s3d
4. Alter 4th Digit
   Alters the 4th digit of the given German ZIP code randomly by $n = 1, 2, 3$ up or down (probability for alter n: $p = n^{\frac{1}{2}}$). Expects a 5 digit German ZIP code string input.
   Shortcut: s4d
5. Alter 5th Digit
   Alters the 5th digit of the given German ZIP code randomly by $n = 1, 2, 3$ up or down (probability for alter n: $p = n^{\frac{1}{2}}$). Expects a 5 digit German ZIP code string input.
   Shortcut: s5d

IMPRINT

Publisher

German Record-Linkage Center
Regensburger Str. 104
D-90478 Nuremberg

Template layout

Christine Weidmann

www.record-linkage.de